OneOS: POSIX + Actors = General-Purpose IoT Platform

Kumseok Jung
University of British Columbia
Electrical and Computer Engineering
Vancouver, BC, Canada
kumseok@ece.ubc.ca

Julien Gascon-Samson
University of British Columbia
Electrical and Computer Engineering
Vancouver, BC, Canada
julien.gascon-samson@ece.ubc.ca

Karthik Pattabiraman
University of British Columbia
Electrical and Computer Engineering
Vancouver, BC, Canada
karthikp@ece.ubc.ca

ABSTRACT

The Internet of Things (IoT) is now a reality. With an increasing number of "smart" devices, a recent interest in *Edge/Fog Computing* has challenged IoT platforms to support general-purpose workloads on arbitrary devices with the same performance and reliability guarantees as the *Cloud*. We present a design of an IoT platform called OneOS, resembling a Distributed Operating System, to provide a single-system image of the entire network of computers. OneOS operates over an abstract machine comprising a grid of high-level language runtimes modeled as Actors. We demonstrate an *evaluation context replacement* technique for mapping the POSIX interface over the networked system to run regular JavaScript and Python programs on OneOS without any modification.

ACM Reference Format:
Kumseok Jung, Julien Gascon-Samson, and Karthik Pattabiraman. 2019. OneOS: POSIX + Actors = General-Purpose IoT Platform. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 MOTIVATION AND APPROACH

To perform more computations at the *Edge*, IoT platforms need to support general-purpose distributed processing in an efficient way. There are two high-level goals of a general-purpose IoT platform: 1) to provide a dependable software infrastructure, 2) to provide a programming environment for a user (e.g. an application developer) to leverage the distributed computing features of the platform.

The majority of IoT middlewares and platforms organize their infrastructure in a 3-tier hierarchical topology comprising interoperating services[3, 5, 8]. While they abstract away the complexity of distributed computing, it comes at the cost of programmability and flexibility; the user must write programs using a particular API and have prior knowledge of the service infrastructure.

We observe that the purpose of an IoT platform is analogous to that of an Operating System (OS). POSIX interfaces, such as pipes and file descriptors, can be mapped onto the networked system to construct an OS-like environment, allowing existing programs to be reused without modification. Reading from a sensor and writing the value into a file should be no more complicated than issuing a single line of shell command: `cat /dev/sensor/1 > sensor-1.log`.

Conference'17, July 2017, Washington, DC, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 SYSTEM DESIGN

Drawing inspiration from work on Distributed OS(DOS)[1, 2, 7], we propose a design of an IoT platform, which we call *Overlay Network Operating System (OneOS)*, operating in the *application layer* over a group of abstract *Actors*[6]. As its model of the underlying machinery is at a higher-level of abstraction than bare metal, it has different responsibilities than a Host OS. In particular, the low-level responsibilities of managing the processor, memory, and I/O are delegated to the local kernels, while OneOS administers higher-level services such as providing a file system interface, scheduling workloads, and coordinating inter-process communication (IPC).

Network Model - We believe that keeping the infrastructure simple is the key to a flexible and future-proof IoT platform, and hence a cluster-like organization is more suitable. The only infrastructural component in OneOS is the Actor middleware, which we call *Runtime*. Its responsibilities are: 1) bootstrapping itself to the network, and 2) running an *Agent* – an Actor abstraction of a program. The *Runtime* knows the location of a *Name Server*, which serves a *Boot Record* pointing to the *Membership Service (MS)* and the *Initialization Manifest (IM)*. *MS* enables the *Runtimes* to make collective decisions through a consensus protocol. *IM* is analogous to the UNIX `init`, describing the core OS services.

Services - *IM* describes 5 *Kernel Agents* that are essential to the system's operation: ① *Publish/Subscribe* service mediates IPC between *Agents*, ② *Scheduler* coordinates the deployment and stateful migration[4] of *Agents*, and ③ *Session* service brokers interactions with an end-user. ④ *File System* provides an indexing mechanism for locating various resources within the network such as files, peripheral devices, and network sockets; adopting the UNIX philosophy that "everything is a file". We decouple the indexing mechanism of the file system and define a separate ⑤ *Storage* service. The *Runtimes* decide via the consensus protocol which of them should run which *Kernel Agent*.

Runtime Model - A *Runtime*'s job is to start a new *Agent* upon receiving a message. The *Runtime* is equipped with high-level language runtimes such as Node.js and CPython, and we assume all *Agents* are written in a supported language. Before instantiating an *Agent* as a child process, the *Runtime* performs an *evaluation context replacement*, in order to interpret the program in the OneOS context instead of the local host context. This is achieved by creating a virtual environment and instrumenting the program on-the-fly to replace system calls like file and I/O read-writes with OneOS API calls, after which the system calls of the child process are redirected to the corresponding *Kernel Agents* and the standard FDs are exposed to the network. We thus take a regular program and run it on OneOS without requiring changes to the original program, seamlessly mapping the POSIX interface over a networked system.

REFERENCES

[1] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. 2009. The Multikernel: A New OS Architecture for Scalable Multicore Systems. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles (SOSP '09)*. ACM, New York, NY, USA, 29–44. https://doi.org/10.1145/1629575.1629579

[2] Sean M Dorward, Rob Pike, David Leo Presotto, Dennis M Ritchie, Howard W Trickey, and Philip Winterbottom. 1997. The Inferno operating system. *Bell Labs Technical Journal* 2, 1 (1997), 5–18.

[3] Eclipse Foundation. 2018. IoT Developer Survey 2018. Retrieved January 24, 2019 from https://www.slideshare.net/kartben/iot-developer-survey-2018

[4] Julien Gascon-Samson, Kumseok Jung, Shivanshu Goyal, Armin Rezaiean-Asel, and Karthik Pattabiraman. 2018. ThingsMigrate: Platform-Independent Migration of Stateful JavaScript IoT Applications. In *32nd European Conference on Object-Oriented Programming (ECOOP 2018) (Leibniz International Proceedings in Informatics (LIPIcs))*, Todd Millstein (Ed.), Vol. 109. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 18:1–18:33. https://doi.org/10.4230/LIPIcs.ECOOP.2018.18

[5] Julien Gascon-Samson, Mohammad Rafiuzzaman, and Karthik Pattabiraman. 2017. ThingsJS: Towards a Flexible and Self-adaptable Middleware for Dynamic and Heterogeneous IoT Environments. In *Proceedings of the 4th Workshop on Middleware and Applications for the Internet of Things (M4IoT '17)*. ACM, New York, NY, USA, 11–16. https://doi.org/10.1145/3152141.3152391

[6] Carl Hewitt. 2017. Actor Model of Computation for Scalable Robust Information Systems. In *Symposium on Logic and Collaboration for Intelligent Applications,*. Stanford, United States. https://hal.archives-ouvertes.fr/hal-01163534

[7] Rob Pike, Dave Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom. 1995. Plan 9 from bell labs. *Computing systems* 8, 2 (1995), 221–254.

[8] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. 2016. Middleware for Internet of Things: A Survey. *IEEE Internet of Things Journal* 3, 1 (Feb 2016), 70–95. https://doi.org/10.1109/JIOT.2015.2498900